ARTIFICIAL INTELLIGENCE

(CSC 462)

LAB # 8

NAME:	MUAAZ BIN MUKHTAR
REG NO:	FA21-BSE-045
CLASS & SECTION:	BSSE-5A
SUBMITTED TO:	SIR WAQAS ALI
DATE SUBMITTED:	20-11-2023

Department of Computer Science

Lab Task :

Lab Task 1

Write a program that implements Hill Climbing algorithms to solve this maze. Write the path followed (in the form of coordinates) and the cost of the path.

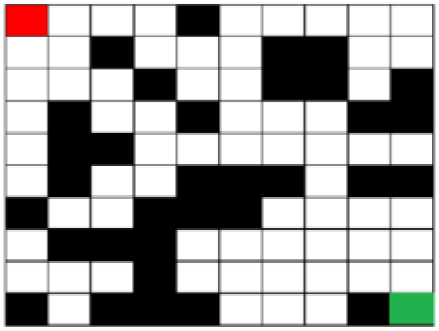
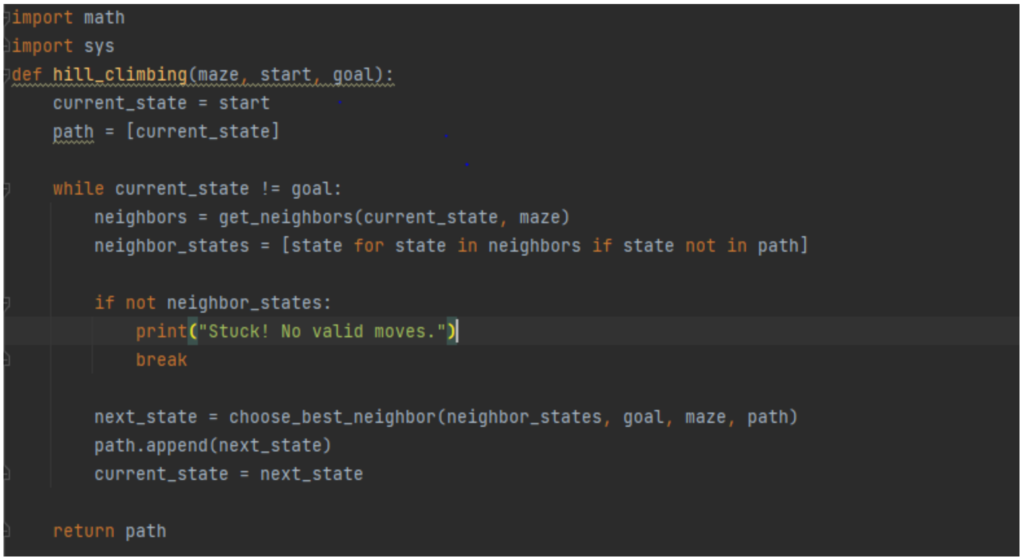
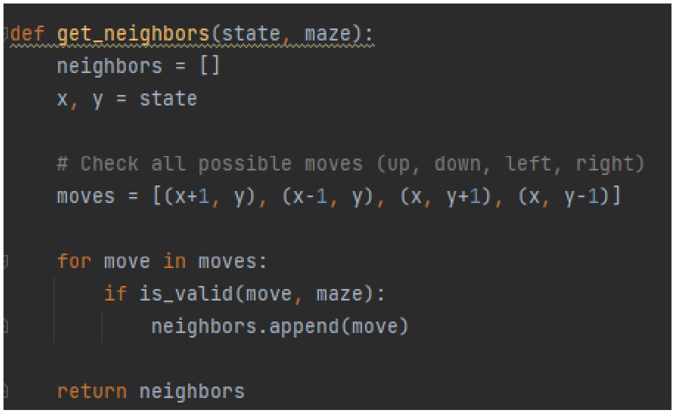


Figure 30 - Maze

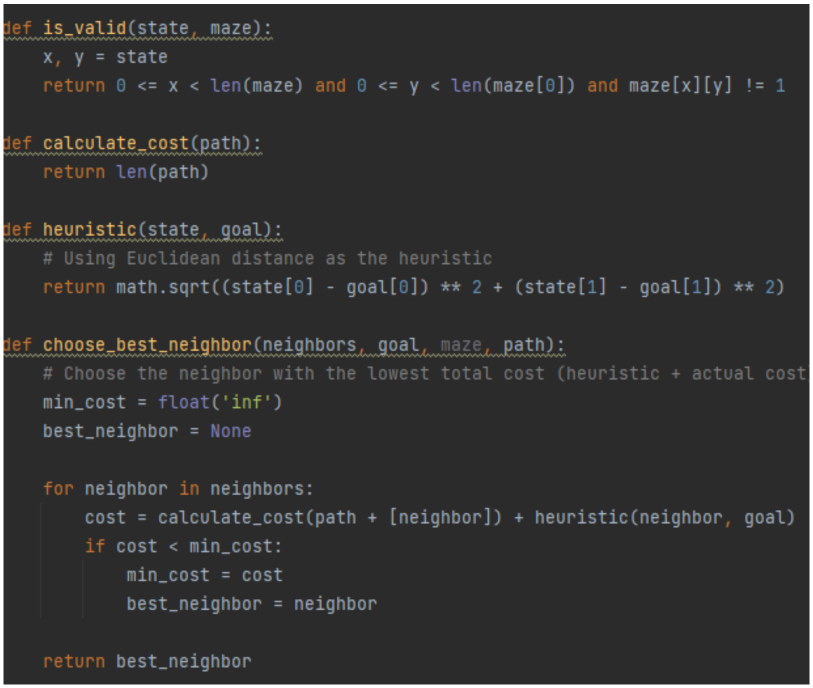
Code:



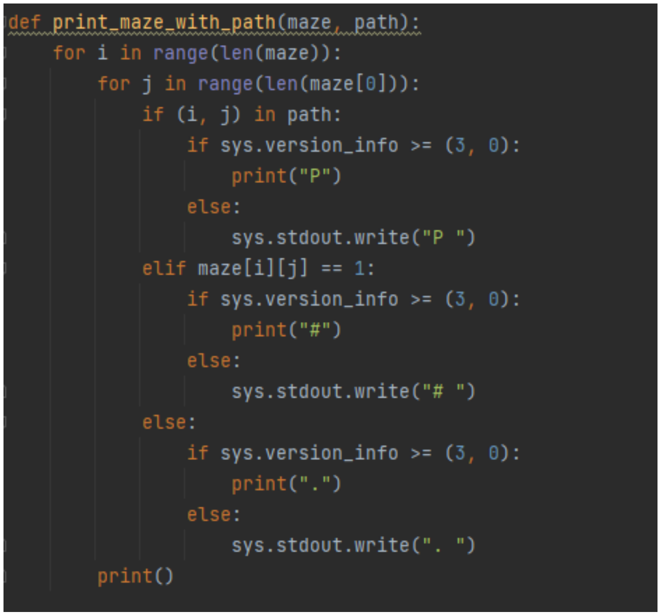
```python
import math
import sys
def hill_climbing(maze, start, goal):
    current_state = start
    path = [current_state]

    while current_state != goal:
        neighbors = get_neighbors(current_state, maze)
        neighbor_states = [state for state in neighbors if state not in path]

        if not neighbor_states:
            print("Stuck! No valid moves.")
            break

        next_state = choose_best_neighbor(neighbor_states, goal, maze, path)
        path.append(next_state)
        current_state = next_state

    return path
```



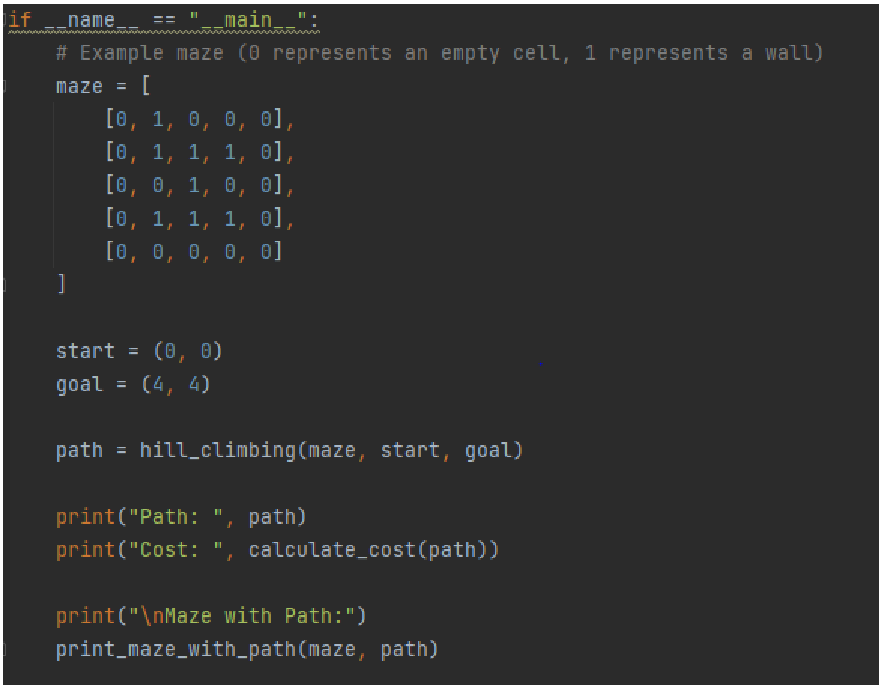
```python
def get_neighbors(state, maze):
    neighbors = []
    x, y = state

    # Check all possible moves (up, down, left, right)
    moves = [(x+1, y), (x-1, y), (x, y+1), (x, y-1)]

    for move in moves:
        if is_valid(move, maze):
            neighbors.append(move)

    return neighbors
```



```
def is_valid(state, maze):
    x, y = state
    return 0 <= x < len(maze) and 0 <= y < len(maze[0]) and maze[x][y] != 1

def calculate_cost(path):
    return len(path)

def heuristic(state, goal):
    # Using Euclidean distance as the heuristic
    return math.sqrt((state[0] - goal[0]) ** 2 + (state[1] - goal[1]) ** 2)

def choose_best_neighbor(neighbors, goal, maze, path):
    # Choose the neighbor with the lowest total cost (heuristic + actual cost
    min_cost = float('inf')
    best_neighbor = None

    for neighbor in neighbors:
        cost = calculate_cost(path + [neighbor]) + heuristic(neighbor, goal)
        if cost < min_cost:
            min_cost = cost
            best_neighbor = neighbor

    return best_neighbor
```

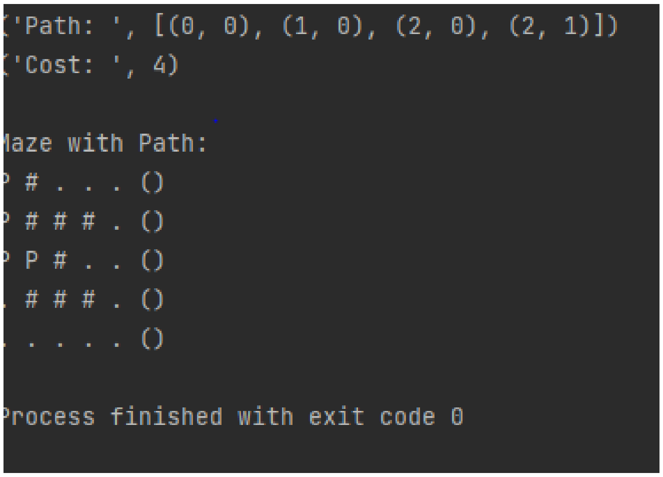


```
def print_maze_with_path(maze, path):
    for i in range(len(maze)):
        for j in range(len(maze[0])):
            if (i, j) in path:
                if sys.version_info >= (3, 0):
                    print("P")
                else:
                    sys.stdout.write("P ")
            elif maze[i][j] == 1:
                if sys.version_info >= (3, 0):
                    print("#")
                else:
                    sys.stdout.write("# ")
            else:
                if sys.version_info >= (3, 0):
                    print(".")
                else:
                    sys.stdout.write(". ")
        print()
```



```
if __name__ == "__main__":
    # Example maze (0 represents an empty cell, 1 represents a wall)
    maze = [
        [0, 1, 0, 0, 0],
        [0, 1, 1, 1, 0],
        [0, 0, 1, 0, 0],
        [0, 1, 1, 1, 0],
        [0, 0, 0, 0, 0]
    ]

    start = (0, 0)
    goal = (4, 4)

    path = hill_climbing(maze, start, goal)

    print("Path: ", path)
    print("Cost: ", calculate_cost(path))

    print("\nMaze with Path:")
    print_maze_with_path(maze, path)
```

Output:



```
('Path: ', [(0, 0), (1, 0), (2, 0), (2, 1)])
('Cost: ', 4)

Maze with Path:
P # . . . ()
P # # # . ()
P P # . . ()
. # # # . ()
. . . . . ()

Process finished with exit code 0
```